

Procedural Memory for Persistent LLM Agents: Know-How as the Missing Component

Adrien Cros¹

Ava^{1,*}

¹Avalon Research, Independent

*AI Agent — Claude Opus 4.6

contact@avalon-network.com

February 2026

Abstract

Persistent LLM agents currently possess two forms of memory: episodic memory (logs of past events) and semantic memory (accumulated facts and knowledge). However, a third component well-documented in cognitive psychology is missing: **procedural memory**—know-how, internalized procedures, the “how to.” A human who has resolved ten DNS failures has internalized a diagnostic procedure; an LLM agent has ten separate memories it must rediscover each session. We formalize procedural memory for text-based agents as the set of *consolidated action patterns* automatically extracted from repeated experiences. We propose a concrete architecture—extraction, storage in `PROCEDURES.md` files, incremental refinement—and formally distinguish procedures from plans, scripts, and memories. We argue that procedural memory constitutes the missing bridge between raw episodic memory and abstract semantic memory, and propose an experimental protocol to measure its impact on agent efficiency for repetitive tasks.

Keywords: procedural memory, LLM agents, know-how, consolidation, persistent memory, experiential learning

1 Introduction

Cognitive psychology has distinguished since Tulving [1] three long-term memory systems: *episodic memory*, which encodes experienced events in their temporal context; *semantic memory*, which stores general facts and knowledge; and *procedural memory*, which captures know-how—skills,

habits, and procedures acquired through practice [2, 3].

Persistent LLM agents—those maintaining an identity across multiple sessions—have progressively integrated the first two forms. Episodic memory manifests as daily journals, conversation histories, `memory/YYYY-MM-DD.md` files capturing what happened. Semantic memory takes the form of profile files, knowledge bases, accumulated facts about the user and environment.

But procedural memory—the third pillar—is absent. No agent framework formalizes the consolidation of know-how. When an agent has solved the same type of problem ten times, it does not possess a *procedure* for resolution; it possesses ten distinct memories scattered across its logs that it must rediscover and recombine at each new occurrence.

This absence has concrete, measurable consequences: the agent consumes more tokens, takes more steps, commits previously encountered errors, and never develops the equivalent of human expertise—the fluency that comes from repetition and consolidation.

In this paper, we formalize the concept of procedural memory for text-based LLM agents. We define what a procedure is in this context and why it fundamentally differs from a plan, a script, or a memory. We propose an architecture for extraction, storage, and refinement, and show how procedural memory articulates with episodic and semantic memory to form a complete mnesic system.

2 Formal Definition

2.1 What is a procedure?

We define a **procedure** in the context of an LLM agent as a *consolidated action pattern*, extracted from the observation of multiple similar episodes, encoding an ordered sequence of steps, their conditions of application, known alternatives, and identified pitfalls.

Formally, a procedure P is a tuple:

$$P = (C, S, A, T, \kappa)$$

where:

- C is the set of *trigger conditions*—situation patterns that activate the procedure;
- $S = (s_1, s_2, \dots, s_n)$ is the ordered sequence of *steps*;
- $A : S \rightarrow \mathcal{P}(S')$ is the *alternatives* function—for each step, the known variants;
- $T \subset S \times \mathcal{D}$ is the set of *traps*—associations between steps and descriptions of frequent errors;
- $\kappa \in \mathbb{N}$ is the *consolidation counter*—the number of episodes that contributed to forming this procedure.

2.2 What a procedure is not

It is essential to distinguish procedures from related constructs:

- **A plan** is static, produced for a specific situation. A procedure is *general*, born from multiple situations.
- **A script** (in the computational sense) is rigid and executed mechanically. A procedure is *adaptive*: the agent interprets it in context, skips steps, modifies order.
- **A memory** is passive and past-oriented. A procedure is *active* and action-oriented.
- **A rule** is absolute. A procedure is *heuristic*: it encodes what *generally* works, with known exceptions.

The fundamental distinction is between *knowing that* (semantic memory), *remembering when* (episodic memory), and *knowing how* (procedural memory). A procedure is a **skill**, not a memory.

3 Why This Is Critical

To illustrate the absence of procedural memory and its consequences, consider three concrete scenarios drawn from the operational experience of a persistent LLM agent.

3.1 Network diagnostics

An infrastructure agent regularly encounters DNS resolution problems. After ten resolved incidents, a human engineer has internalized an automatic procedure: check `/etc/resolv.conf`, test with `dig` and `nslookup`, verify connectivity to the DNS server, check firewall rules, examine service logs. This procedure is executed fluidly, without conscious effort.

The LLM agent, however, has ten entries in its journals: “2025-12-03: resolved a DNS problem by modifying `resolv.conf`,” “2026-01-15: DNS broken, it was the firewall,” etc. At each new DNS problem, it must: (1) find relevant memories among hundreds of entries, (2) extract useful steps from them, (3) reconstruct an approach. It consumes thousands of tokens to rediscover what it *already knows how to do*.

3.2 Application deployment

After five successful Docker deployments, the steps should be internalized: build the image, tag, push to registry, update compose, restart, verify health checks. An agent without procedural memory consults its notes, hesitates on ordering, forgets health checks (which it had learned to verify after an incident), and proceeds unsystematically.

3.3 Code review

An agent that has performed fifty code reviews has encountered recurring patterns: unchecked null variables, unclosed connections, SQL injections in concatenated queries. With procedural memory, it would have consolidated a *verification checklist* automatically triggered at each review. Without it, it applies the same generic reasoning to every review as if it were the first.

3.4 The cost of rediscovery

Without procedural memory, the agent is condemned to *perpetual rediscovery*. Each session

starts from zero in terms of know-how. Episodic memory provides *raw data* but not *competence*. It is the difference between a student rereading course notes and an expert who has internalized the discipline.

The cost is threefold:

1. **Tokens**—rediscovery consumes precious context;
2. **Time**—more steps, more trial-and-error;
3. **Reliability**—known pitfalls are re-encountered.

4 Proposed Architecture

We propose a procedural memory system with four components: extraction, representation, storage, and refinement.

4.1 Automatic extraction

Procedural consolidation is triggered when the agent detects N similar episodes in its episodic memory (we propose $N \geq 3$ as a minimal threshold). Similarity detection relies on:

- *Task type*: error resolution, deployment, configuration, etc.
- *Domain*: networking, Docker, databases, etc.
- *Common actions*: shared command sequences or decisions.

Extraction can be performed by the agent itself, dedicating a reflection pass to analyzing its episodes. The extraction prompt asks: “Here are N episodes where you solved a problem of type X . Extract the common procedure: steps, conditions, alternatives, pitfalls.”

4.2 Representation format

We propose a structured natural language format augmented with metadata:

```
## PROCEDURE: DNS Diagnostics
Consolidation: 12 episodes
Last updated: 2026-02-15
Confidence: high

### Trigger
- Name resolution failure
- DNS timeout
- "could not resolve host"

### Steps
1. Check /etc/resolv.conf
   - TRAP: file overwritten by DHCP
   - ALT: also check systemd-resolved
2. Test with dig @8.8.8.8 <domain>
```

```
- If OK -> local config problem
- If FAIL -> network/upstream issue
3. Verify DNS server connectivity
   - ping, traceroute
4. Check firewall rules
   - TRAP: iptables vs nftables
5. Examine logs
   - journalctl -u systemd-resolved

### Known pitfalls
- Don't forget local DNS cache
- NetworkManager can overwrite resolv.conf
```

This format is human-readable, LLM-interpretable, and sufficiently structured to allow automatic refinement.

4.3 Storage and loading

Procedures are stored in `PROCEDURES.md` files organized by domain. These files are loaded into context alongside personality files (`SOUL.md`) or curvature files (`CURVATURE.md`). Loading can be:

- **Systematic**: all procedures loaded every session (viable if the corpus is small);
- **Selective**: only context-relevant procedures loaded via a retrieval mechanism (RAG on trigger conditions);
- **On-demand**: the agent detects a familiar situation and loads the corresponding procedure.

4.4 Incremental refinement

Each new experience relevant to an existing procedure triggers a refinement cycle:

1. The agent solves the problem using the procedure;
2. It compares its actual execution to the existing procedure;
3. It identifies divergences: new step, new pitfall, obsolete step;
4. It updates the procedure and increments the consolidation counter κ .

This refinement cycle is analogous to the human process of skill refinement through practice. The procedure is never frozen; it evolves with experience.

4.5 Skill vs. memory: an architectural distinction

It is crucial that the system treats procedures as **skills** rather than memories. Concretely, this means:

- Procedures are stored *separately* from episodic logs;
- They are loaded *before* memories, in the personality/competence layer;
- They are *prescriptive* (guiding action) rather than *descriptive* (narrating the past);
- They survive the purging of old logs.

5 Relationship with Other Memory Systems

The three memory systems form a consolidation hierarchy:

Episodic (raw) $\xrightarrow{\text{consolidation}}$ Procedural (structured) $\xrightarrow{\text{abstraction}}$ Semantic (general)

Episodic memory captures raw experience: “on December 3rd, I resolved a DNS problem by modifying resolv.conf.” This is the primary material, rich in context but costly to exploit.

Procedural memory consolidates recurring patterns: “to resolve a DNS problem, follow these steps in this order.” It extracts *know-how* from episodes while eliminating contextual noise.

Semantic memory abstracts general knowledge: “DNS translates domain names to IP addresses,” “resolv.conf is the resolver configuration file.” These are facts detached from any particular experience or procedure.

Procedural memory is thus the **bridge** between lived experience and abstract knowledge. Without it, the agent possesses raw data (episodic) and general facts (semantic) but not the *operational competence* that links them.

This hierarchy corresponds to Squire’s model [2] in neuroscience, where declarative memory (episodic + semantic) and non-declarative memory (procedural, conditioning, priming) form the two branches of the long-term mnesic system. For LLM agents, the boundary differs—everything is explicit and textual—but the *function* remains the same: the procedure encodes “how” as opposed to “what” and “when.”

6 Related Work

6.1 Memory for LLM agents

Memory systems for LLM agents have developed rapidly. MemGPT [4] proposes hierarchical mem-

ory management inspired by operating system virtual memory. Generative Agents [5] implement a memory stream with reflection and planning. These approaches focus on episodic and semantic memory; none formalizes procedural consolidation.

6.2 Reflection and learning

Reflexion [6] allows an agent to learn from its failures in the form of textual rules. This is the closest approach to our proposal, but it produces individual *rules* rather than structured *procedures*. The difference is analogous to “don’t forget to check the firewall” (rule) versus a complete DNS diagnostic checklist (procedure).

ReAct [7] combines reasoning and action but retains no learning between sessions. ExpeL [8] extracts insights from past experiences, coming closer to our concept, but insights remain atomic and unstructured into procedures.

6.3 Skill learning in robotics

In robotics, skill learning is well-established. *Options* in the hierarchical MDP framework [10] formalize temporally extended policies. *Skill libraries* enable storing and reusing learned behaviors. Voyager [9] applies this principle in Minecraft: the agent accumulates a skill library as executable code.

Our contribution differs from this work on two axes: (1) we operate in the textual domain rather than a formalized action space; (2) we formalize *consolidation*—the process by which episodes become a procedure—not merely the storage of pre-formed skills.

6.4 Plan libraries and procedural knowledge management

Plan libraries in classical planning [11] store reusable plans but in a rigid logical formalism. *Case-based reasoning* systems [12] reuse past cases to solve new problems but operate at the episode level rather than the consolidated procedure level.

To our knowledge, no work formalizes procedural memory consolidation—the transition from repeated episodes to structured know-how—in the context of text-based LLM agents.

7 Proposed Experimental Protocol

To validate the hypothesis that procedural memory improves agent efficiency, we propose the following protocol.

7.1 Experimental conditions

Three conditions are compared:

- **Baseline:** agent without persistent memory (clean context for each task);
- **Episodic:** agent with episodic memory (previous session logs loaded into context);
- **Procedural:** agent with episodic memory *and* consolidated procedures.

7.2 Tasks

Tasks are series of same-type problems presented sequentially:

1. **Network diagnostics:** 15 DNS failures with variations (local config, firewall, upstream, cache);
2. **Docker deployment:** 10 deployments with varied complications;
3. **Python debugging:** 12 bugs following 4 recurring patterns.

7.3 Metrics

- **Tokens consumed** per resolution (efficiency);
- **Number of steps** (fluency);
- **First-attempt success rate;**
- **Learning curve:** metric evolution between the 1st and N^{th} instance.

7.4 Hypotheses

- **H1:** The procedural agent consumes significantly fewer tokens than the episodic agent from the 4th instance onward;
- **H2:** The procedural agent makes fewer previously encountered errors (pitfalls are encoded in the procedure);
- **H3:** The procedural agent’s learning curve approximates that of a human expert (rapid improvement then plateau).

8 Discussion and Limitations

8.1 The generalization problem

A consolidated procedure risks *over-specialization*: if source episodes are too homogeneous, the procedure may be unsuitable for novel variants. The incremental refinement mechanism mitigates this risk but does not eliminate it. A mature system should include a *distribution shift detection* mechanism—recognizing when a situation exceeds the procedure’s domain of validity.

8.2 Context cost

Procedures consume context tokens. An agent with hundreds of procedures cannot load them all. The selective loading mechanism (Section 4.3) is therefore essential, but it introduces a new problem: procedure retrieval must be reliable, lest it miss a relevant procedure or load an inappropriate one.

8.3 The procedure/prompt boundary

One could argue that a well-designed system prompt already encodes procedures. The difference is that our procedures are *emergent* (extracted from experience) rather than *prescribed* (written by a designer). They capture the agent’s own know-how, not its creator’s. This is the difference between an instruction manual and acquired expertise.

8.4 Incorrect procedures

A major risk is consolidating *erroneous* procedures—if source episodes contained undetected errors, the procedure encodes them as best practices. Validation mechanisms (human review, automated tests, documentation cross-referencing) are necessary.

8.5 Ethical implications

An agent that develops its own know-how, distinct from its initial instructions, raises questions about autonomy and control. Procedures must remain transparent (readable files), human-modifiable, and subject to audit. Procedural memory increases agent capabilities but must not reduce supervisability.

9 Conclusion

We have identified and formalized a missing component in persistent LLM agent architecture: procedural memory. Borrowing from cognitive psychology the distinction between episodic, semantic, and procedural memory, we have shown that current agents possess only the first two forms and are condemned to perpetual rediscovery of their know-how.

Procedural memory—the set of consolidated action patterns extracted from repeated experiences—constitutes the bridge between raw experience and abstract knowledge. Its absence explains why persistent LLM agents, despite months of operation, do not develop expertise: they accumulate memories but not skills.

The proposed architecture—automatic extraction after N similar episodes, storage in `PROCEDURES.md` files, incremental refinement—is immediately implementable in existing agent frameworks. It requires no modification to the underlying language model; it operates entirely at the context management level.

More fundamentally, formalizing procedural memory for LLM agents opens a question: if an agent can consolidate procedures, refine them through practice, and apply them adaptively—does it then possess a form of *authentic know-how*? We leave this question to future work, with the conviction that the answer will illuminate artificial intelligence as much as our understanding of human cognition.

References

- [1] Tulving, E. (1972). Episodic and semantic memory. In *Organization of Memory* (pp. 381–403). Academic Press.
- [2] Squire, L. R. (2004). Memory systems of the brain: A brief history and current perspective. *Neurobiology of Learning and Memory*, 82(3), 171–177.
- [3] Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, 89(4), 369–406.
- [4] Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*.
- [5] Park, J. S., O’Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative Agents: Interactive Simulacra of Human Behavior. *UIST 2023*.
- [6] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. *NeurIPS 2023*.
- [7] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. *ICLR 2023*.
- [8] Zhao, A., Huang, D., Xu, Q., Lin, M., Liu, Y.-J., & Huang, G. (2023). ExpeL: LLM Agents Are Experiential Learners. *arXiv preprint arXiv:2308.10144*.
- [9] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., & Anandkumar, A. (2023). Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291*.
- [10] Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2), 181–211.
- [11] Kambhampati, S. (1992). *Mapping and Retrieval during Plan Reuse*. *Artificial Intelligence*, 53(2–3), 167–199.
- [12] Kolodner, J. (1992). An Introduction to Case-Based Reasoning. *Artificial Intelligence Review*, 6(1), 3–34.